Protocolo Práctico: Introducción al Constrained Based Modeling con COBRApy

Miguel Ponce de León

Parte 0: Accediendo al espacio de trabajo

- Abrir una terminal a iniciar una sesión de jupyter notebook:
  - ***jupyter notebook***

Cobrapy es un módulo o biblioteca dedicada al CBM que fue escrita en lenguaje python (existe una versión similar para matlab).

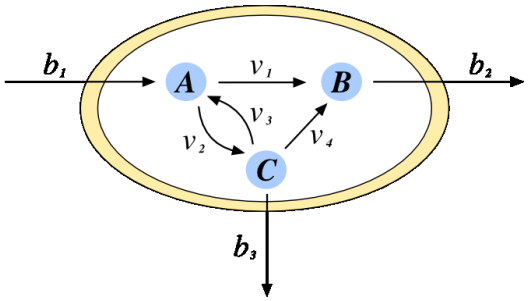
- Documentación: https://cobrapy.readthedocs.org/en/latest/
- Referencia: http://www.biomedcentral.com/1752-0509/7/74
- Repositorio: https://github.com/opencobra/cobrapy

Parte 1 – Introducción al Constraind-based Modeling con cobrapy

1.1 – Creando nuestro primer modelo metabólico con cobra (ver esquema)

Crear un modelo que incluye crear las reacciones y los metabolitos representados en el esquema de la figura 2.

a)



Esquema

a)

$$\begin{bmatrix} \frac{dA}{dt} \\ \frac{dB}{dt} \\ \frac{dC}{dt} \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & -1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 & -1 \end{bmatrix}}_{N} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}_{v}$$

Modelo

Figura 2. Modelo metabólico de juguete compuesto por tres metabolito (A, B y C), cuatro reacciones internas (v1-v4) y tres flujos de intercambio (b1-b3). a) representación gráfica del modelo; b) representación matricial de las ecuaciones de balance. N tambien llamada S es la matriz estequiométrica.

```
# Importamos las módulos necesarios del paquete cobra
import cobra
import cobra.core
from cobra.core import Model, Reaction, Metabolite

# Creamos un modelo y lo llamamos Toymodel
model = Model('Toymodel')

# Creamos los metabolitos y definimo su compartimento
A = Metabolite("A")
A.compartment = 'cytosol'
B = Metabolite("B")
B.compartment = 'cytosol'
C = Metabolite("C")
C.compartment = 'cytosol'

# creamos una lista de metabolitos y los agregamos al sistema
the_metabolites = [A, B, C]
model.add_metabolites(the_metabolites)

# Creamos las reacciones
b1 = Reaction("b1")
b2 = Reaction("b2")
```

```
b3 = Reaction("b3")

v1 = Reaction("v1")
v2 = Reaction("v2")
v3 = Reaction("v3")
v4 = Reaction("v4")

# agregamos los metabolitos a las reacciones
# pasando un diccionario (hash) con las variables metabolito como clave
# y el coeficiente estequiometrico como valores (no importa el orden)
b1.add_metabolites({A: 1})
b2.add_metabolites({B: -1})
b3.add_metabolites({C: -1})

v1.add_metabolites({A:-1, B:1})
v2.add_metabolites({A:-1, C:1})
v3.add_metabolites({A:1, C:-1})
```

Ejercicio 1: agregar los metabolitos correspondientes a la reacción v4

```
# creamos una lista con las variables reaccion y agregamos al modelo
the_reactions = [v1,v2,v3,v4,b1,b2,b3]
model.add_reactions(the_reactions)

# para poder ver la matriz estequiometrica, primero
# se debe convertir el modelo a arrays
abm = model.to_array_based_model()

# mostrar por pantalla la matriz estequiométrica y compararla con la del
diagrama
print abm.S.todense()
```

1.2 – Escritura del modelo en formato SBML (opcional)

```
# guardaremos el modelo en formato SBML
# importamos los modulos y las funciones ncesarias
import cobra.io
from cobra.io import write_sbml_model

# guardamos el modelo en formato SBML
write_sbml_model(model,'toymodel.xml',use_fbc_package=False)
```

Ejercicio 2 (opcional): abrir el fichero SBML toymodel.xml en geany u otro editor de texto para ver la estructura

1.3 – Optimización – calculo de una distribución de flujos óptima

El problema es encontrar encontrar un conjunto de valor de flujo (o velocidades) para cada reacción de la red, que sea compatible con las restricciones del análisis estequiométrico (i.e. estado estacionario, reversibilidades y cotas) y que sea óptima (maxima o minimal) respecto a algún objetivo definido.

```
# Antes de optimizar debemos establecer la condiciones de crecimiento
# es decir las velocidades máximas de los flujos de entrada
model.reactions.b1.upper_bound = 1

# Luego indicamos la velocidad que deseamos optimizar (i.e. nuestro
# objetivo) que para nuestro caso será optimizar el flujo de b2
# Para esto asignamos 1 al coeficiente objetivo de b2
model.reactions.b2.objective_coefficient = 1

# Finalmente optimizamos el modelo!
model.optimize()

# Para ver la solución
print "Solucion optima! b2=", model.solution
print
print model.solution.x_dict
print
# En forma alternativa
for k in model.solution.x_dict:
    print k + ":",model.solution.x_dict[k]
```

Ejercicio 3 (opcional): simular un knockout sobre v1

```
# simulamos un knockout sobre la reaccion v1 fijando su valor a cero
# utilizando el atributo upper_bound
model.reactions.v1.upper_bound = 0

# volvemos a optimizar y comparamos la dos soluciones
# puede ser de ayuda dibujar el esquema y poner valores a los flujos
# para poder comparar ambos casos
```

Parte 2: Genome-Scale-Modeling

En esta sección emplearemos un modelo metabólico de E. coli reconstruido a escala genómica denominado *iJO1366*.

2.1 – Lectura de fichero SBML e inspección general del modelo

importar cobra.io si no fue importada (ver 1.2).
Lectura del fichero SBML (modelo metabólico a escala genómica de Escherichia coli). Para esto deberán tener descargado el fichero **`iJO1366.xml`**

```
import cobra
from cobra.io import *
sbml_fname = 'iJO1366.xml'
# (O en su defecto la ruta a donde se hayan descargado el archivo iJO1366.xml)
model = read_sbml_model(sbml_fname)
```

Para verificar que el modelo ha sido cargado de forma correcta podéis probar el siguiente comando:

```
print model
```

El ordenador debe imprimir en pantalla: **iJO1366**
Extraer propiedades generales del modelo empleando las siguiente líneas:

- Listar reacciones: **`model.reactions`**
- Número total de reacciones: **`print len(model.reactions)`**
- Seleccionar varias reacciones y ver: su nombre, formula, gen/es codificante/s, reversibilidad y las cotas. Por ejemplo para la NADH16pp:

```
print 'Name:',model.reactions.NADH16pp.name
print model.reactions.NADH16pp.reaction
print ' '.join(map(str,model.reactions.NADH16pp.genes))
print 'UB:',model.reactions.NADH16pp.upper_bound
print 'LB',model.reactions.NADH16pp.lower_bound
```

http://ecocyc.org/ECOLI/NEW-IMAGE?type=NIL&object=EG12084&redirect=T

- Listar metabolitos: **`model.metabolites`**
- Número total de metabolitos: **`print len(model.metabolites)`**
- Seleccionar varios metabolitos y ver: su nombre, el compartimiento y las reacciones en las que participa. Por ejemplo para la g6p_c:

```
print model.metabolites.g6p_c.name
print model.metabolites.g6p_c.compartment

for r in model.metabolites.g6p_c.reactions:
    print r.name + ':',r.reaction
```

2.2 – Cálculo del modo de flujos óptimo, utilizando Flux Balance Analysis (FBA)

Realizar la optimización (maximizar) la producción de biomasa. La condición de borde que el modelo tiene por defecto, permite una entrada ilimitada de oxigeno pero hay un limite en la entrada de glucosa fijado en -10[1]. Verificar lo anterior mirando los atributos *lower_bound* de las reacciones de intercambio: EX_glc_e_ y EX_o2_e_.

```
# Realizar la optimización:
model.optimize()

# Valor de la función objetivo (biomasa) en el óptimo:
model.solution.f

# Ver el vector solución:
model.solution.x

# Ver el vector solución, con los nombres correspondientes a cada variable (reacciones):
model.solution.x_dict

# También se puede inspeccionar el valor del flujo en elpropio objeto reacción:
model.reactions.PFK.x
```

Identificar en el listado generado anteriormente: el valor que toman los flujos de intercambio de la glucosa (EX_glc_e_) y el oxígeno (EX_o2_e_) en la solución óptima calculada y compararlo con el valor de los lower_bound de ambas reacciones.

2.3 - Visualizando Modos de Flujo Óptimos

Para poder realizar esta parte vamos a emplear una herramienta de visualización web Pero esta herramienta requiere que le demos un fichero con las reacciones del modelo y los valores de los flujos de la solución que deseamos visualizar. De forma que hace falta guardar los resultados en un fichero de texto en formato json y que nos descarguemos el fichero a nuestro ordenador personal.

```
# Para facilitar la visualización vamos a emplear un método alternativo de
# optimización que se llama pFBA. Basicamente es un FBA que luego elimina #
# todos los ciclos que aparezcan en una solución. # para esto se debe
# import la función optimize_minimal_flux que se encuentra en el módulo
# parsimonious
from cobra.flux_analysis.parsimonious import optimize_minimal_flux
optimize_minimal_flux(model)
```

1 Por convención el signo de un flujo de intercambio sobre un metabolito externo, es negativo si el sistema consume el metabolito externo y positivo si lo produce.

```
# Para crear y cuardar el fichero hay que seguir los siguiente pasos
# primero importamos el módulo json
import json
# luego guardamos la solución en una variable de tipo diccionario
wt_solution = model.solution.x_dict
# abrimos un fichero de nombre wt_solution.json en modo escritura
f = open('wt_solution.json','w')
# escribimos el en el fichero y cerramos el fichero
json.dump(wt_solution,f)
f.close()
```

Accedemos a la herramienta de visualización web a través de la siguiente URL:
http://escher.github.io/

Parte 3: *In-silico* knockout experimets

3.1 – Knockout *in silico*

Un experimento de *knockout in silico* consiste en fijar a cero el flujo a través de todas las reacciones que cataliza el gen para el cual estamos simulando el *knockout*.
Ejemplo 1: el gen que codifica para la actividad Citrato Sintasa es el gen **b0720.** Para poder ver dicha asociación basta ver los siguiente:

```
# seleccionamos el gene
gene =  model.genes.b0720
# seleccionamos las reacciones asociadas a dicho gen
# imprimimos las reacciones asociadas a b0720
for r in gene.reactions:
    print "ID:%s | Nombre: %s" % (r.id,r.name)

# Lo mismo comenzando desde la reacción.
reaction = model.reactions.CS
for g in reaction.genes:
    print "ID:%s" % g.id
```

Entonces simular un *knockout in silico* de **b0720** equivale a fijar a 0 la reacción **CS** para lo cual utilizamos los atributos upper_bound (ya que el lower_bound es 0 porque la reacción es irreversible):

```
model.reactions.CS.upper_bound = 0
# Buscamos el modo de flujo que maximiza el crecimiento
model.optimize()
# miramos el valor del crecimiento predicho, que sucede?
model.solution
```

¿El experimento *in silico* que nos indica? ¿El *knockout* para **b0720** es letal?
Compara con el resultado *in vivo* en esta base de datos:
https://ecocyc.org/gene?orgid=ECOLI&id=EG10402

3.2 – Knockout *in silico:* Large Experiment

Cobra tiene implementada una función para realizar los experimentos de *knockout*. En esta sección emplearemos dicha función para realizar un experimento de *knockout* sobre todos los genes que incluye el modelo y esto lo compararemos con resultados *in vivo*
Primero se requiere cargar los siguiente módulos:

```
import json
from cobra.flux_analysis import single_gene_deletion

# Leemos el modelo nuevamente (para resetear los parámetros)
model = cobra.io.read_sbml_model(sbml_fname)

# experimento in-silico (tarda unos minutos)
knockout_exp,stat = single_gene_deletion(model)

knockout_exp[0]

# crear un "set" con los nombres de todos los genes del modelo
all_genes = set([g.id for g in model.genes])

# Lectura de la lista de genes letales in vivo
fname = 'm9_invivo_lethals.json'
invivo_lethals = json.load(open(fname))

# aplicamos una operación de conjuntos (set) para obtener los genes no
# letales in-vivo
invivo_non_lethals = all_genes.difference(invivo_lethals)

# Obtención de letales in silico filtrando lo genes cuyo knockout causo una
# caída en la producción de biomasa por debajo de un umbrar (< 5% del
optimo)
insilico_lethals = set([k for k,v in knockout_exp[0].items() if v<0.05])

# Obtención de no letales in silico que son los que si predicen crecimiento
insilico_non_lethals = set([k for k,v in knockout_exp[0].items() if
v>=0.05])
```

```
TP = 1.0 * len(insilico_non_lethals.intersection(invivo_non_lethals))
TN =  1.0 * len(insilico_lethals.intersection(invivo_lethals))
FP = 1.0 * len(insilico_non_lethals.intersection(invivo_lethals))
FN = 1.0 * len(insilico_lethals.intersection(invivo_non_lethals))

P = TP + FN
N = TN + FP
```

EJERCICIO A – Crecimiento en condiciones anaeróbicas

El objetivo de este ejercicio es comparar las distribuciones de flujo predichas mediante FBA en dos condiciones de crecimientos diferentes:

- *Medio mínimo con glucosa aeróbico (es el caso que se vio en clase)*
- *Medio mínimo con glucosa anaeróbico*

Para poder simular el crecimiento anaeróbico debemos modificar la cota correspondiente al flujo de intercambio del oxígeno. Por convención los flujos de intercambio se consideran negativos cuando el sistema consume un metabolito del exterior (fuente) y positivos cuando el sistema produce o excreta un producto (sumidero). De forma que si queremos simular un medio **anaeróbico debemos** restringir la entrara de oxígeno para lo cual debemos **fijar en cero la cota interior del flujo de intercambio de oxígeno.** Luego del cambio optimizamos el modelo para determinar la solución que maximiza la producción de biomasa en las nuevas condiciones y finalmente guardamos la solución en un fichero json.

```
from cobra.flux_analysis.parsimonious import optimize_minimal_flux

# antes de hacer el experimento se recomienda volver a leer el modelo
model = cobra.io.read_sbml_model(sbml_fname)

model.reactions.EX_o2_e_.lower_bound = 0
optimize_minimal_flux(model)

anaerobic_solution = model.solution.x_dict
f = open('anaerobic_solution.json','w')
json.dump(anaerobic_solution,f)
f.close()
```

Descargar el fichero y construir el mapa con escher. Comparar el mapa obtenido con el mapa que se realizó en el punto 2.3. Discutir la diferencia haciendo énfasis en las productos de excreción que se producen en cada una de las dos condiciones.

EJERCICIO B – Single knockout experiment.

El gen que codifica para la actividad Enolasa es el gen b2779. Se desea simular un knockout in-silico sobre el gen y de las predicciones del modelo

```
# antes de hacer el experimento se recomienda volver a leer el modelo
model = cobra.io.read_sbml_model(sbml_fname)
```

1. Repetir experimento realizado en 3.1 pero empleando el gen b2779
2. Comparar con el resultado con el experimento *in-vivo* (consultar en la base de datos Ecocyc)
3. Discutir y comentar el resultado

EJERCICIO C – Large-scale knockout experiment.

Emplear los resultado obtenidos en el punto 3.2, i.e. los valores de verdaderos y falsos positivos y negativos para completar la siguiente tabla:

		In vivo	
		No letal	Letal
In silico	No letal	**TP**	**FP**
	Letal	**FN**	**TN**

Con los resultado obtenidos calcular:

- **Especificad** (specificity) del método
- **Sensibilidad** (sensitivity) del método
- **Exactitud** (accuracy) el método

Si no saben las fórmulas podéis acceder a las definiciones mediante el siguiente link:
http://en.wikipedia.org/wiki/Sensitivity_and_specificity

Comentar los resultados obtenidos haciendo énfasis en que tipo de errores comete más el modelo y cual pude ser la causa.